EP2322261

OPIC
Office de la propriété intellectuelle du Canada

CIPO
Canadian Intellectual Property Office

(12) (19) (CA) **Demande-Application**

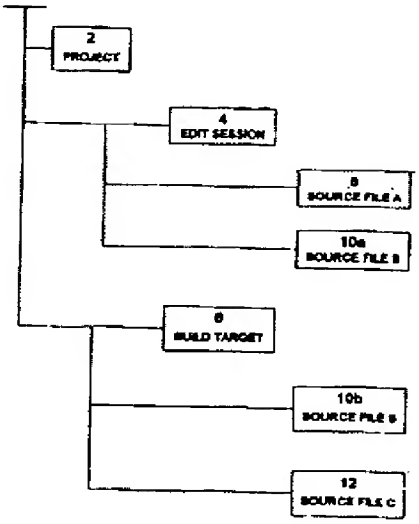
(21) (A1) **2,201,276**
(22) 1997/03/27
(43) 1998/09/27

(72) SLUIMAN, Harm, CA
(72) STARKEY, Michael, CA
(71) IBM CANADA LIMITED - IBM CANADA LIMITEE, CA

(51) Int.Cl.[6] G06F 17/30

(54) **VUES HIERARCHIQUES INDIRECTES POUR LA GESTION D'APPLICATIONS LOGICIELLES**

(54) **INDIRECT HIERARCHICAL VIEWS FOR SOFTWARE APPLICATION MANAGEMENT**



(57) Technique de gestion de multiples vues contextuelles de fiches de codes sources. Dans un système de fichiers, différentes tâches exigent le groupage de fichiers de codes sources dans différents répertoires hiérarchiques. Lorsque cette manipulation vise en partie ou intégralement les fichiers physiques eux-mêmes, les répertoires peuvent se mêler, et l'utilisateur suivant peut avoir du mal à accéder aux fichiers de codes sources dont il a besoin. Suivant présente technique, l'emplacement physique des fichiers est indépendant de toutes les vues contextuelles et maintenu dans une hiérarchie visualisable distincte. Tous les éléments des hiérarchies de vues contextuelles contiennent seulement des références à l'emplacement physique des fichiers de codes sources; ainsi, les modifications de ces articles, p. ex. suppression, déplacement et copie, ne touchent que les références elles-mêmes. Un ensemble distinct d'opérations peut être utilisé pour la modification ou le maintien explicite des fichiers physiques de codes sources.

(57) The invention provides a technique for managing multiple contextual views of source code files. In a file system, different tasks demand that source code files be grouped in different hierarchical directories. When some or all of this manipulation is of the physical files themselves, the directories can become confused, and it may be difficult for the next user to access needed source code files. In the technique of the invention, the physical location of the files is independent of all contextual views, and is maintained in a separate viewable hierarchy. All items in contextual view hierarchies contain only references to the physical location of the source code files, and so modifications of those items, such as delete, move and copy, only alter the references themselves. A separate set of operations can be used for explicitly maintaining or modifying the physical source code files.

CA9-97-005

INDIRECT HIERARCHICAL VIEWS FOR SOFTWARE APPLICATION MANAGEMENT

Abstract

The invention provides a technique for managing multiple contextual views of source code files. In a file system, different tasks demand that source code files be grouped in different hierarchical directories. When some or all of this manipulation is of the physical files themselves, the directories can become confused, and it may be difficult for the next user to access needed source code files. In the technique of the invention, the physical location of the files is independent of all contextual views, and is maintained in a separate viewable hierarchy. All items in contextual view hierarchies contain only references to the physical location of the source code files, and so modifications of those items, such as delete, move and copy, only alter the references themselves. A separate set of operations can be used for explicitly maintaining or modifying the physical source code files.

2201276

INDIRECT HIERARCHICAL VIEWS FOR SOFTWARE APPLICATION MANAGEMENT

Field of the Invention

The present invention is directed to information handling systems, and in particular, to a general use software management technique.

Background of the Invention

Applications usually consist of a number of independent source files maintained in a file system Software developers can work with the source code contained in these in a number of ways. A source code segment can be edited as part of a larger coding effort, and/or compiled into an executable or even into several different executables. Thus, a single source file can be used in several contexts, and any of these context can also be part of a body of multiple overlapping contexts.

However, the single source file physically exists in only one location.

When developing software, one of the first tasks of the developer or manager is often to determine the nature and location of the source file contents in the application being used. This is frequently done by using disk directories, on an ad hoc basis, as an organizational vehicle. Some common examples are directory structures to reflect project builds or makes, directory structures to reflect source version hierarchies, and project and department organizations.

The content of a request may require a change in the containment hierarchies of the directory structure. Therefore, a common approach is to initially organize the directories for the most frequently-used context, and devise dedicated software "tools" to copy or rearrange the directories to support the context required by other task demands, as they arise.

Some file systems provide mechanisms for creating "shadow objects" copies of the original source files, and hierarchies. These are then maintained as file system directories in the ordinary manner.

All of this leaves the structure of the source files and the directories in various states typically understood only by the user that organized it. Often this is simply the last requester.

As the complexity and/or size of a project grows, this problem becomes more of an exposure. Steps must be taken to correct it, but there is currently no known management system providing a flexible point of control, particularly over directory structures that have become standardized and unchangeable.

Summary of the Invention

It is therefore an object of the present invention to provide a management system to separate the different hierarchical "views" of source code information taken by requests, from the physical location of the source code and from the limitations of the file system structure generally.

Accordingly, the present invention provides a mechanism and process for managing source code files in file directory hierarchies in a file data processing system. A first hierarchy with at least one branch containing objects representing the physical location of the source code files is created, as are additional hierarchies having branches representing contextual views of items from the source code files. The items are references to individual source code files.

Brief Description of the Drawings

Embodiments of the invention will now be described in detail in association with the accompanying drawings in which:

Figure 1 is a simple multiple schematic view of file management;

Figure 2 is a hierarchy similar to Figure 1, of context and physical views, arranged according

to the invention; and

Figure 3 is a window view showing the hierarchical context and physical views, according to a preferred embodiment of the invention.

Detailed Description of the Preferred Embodiments

User tasks or contexts can be described as a set of serial and/or parallel structure. In particular, a large number of contexts can be described by containment relationships. Containment relationships map very cleanly to hierarchical tree views with the root, or a top node, being the highest parent container.

Figure 1 illustrates a schematic or tree view of a project 2 containing an edit session 4 and a build target or compilation unit 6. Each of these branches contains source files 8, 10 and 12 that are either being edited or compiled.

Source file B, generally designated as 10 appears twice, both under the edit session 4 and under the build target 6 branches. Depending on the nature of the user, this could mean that there are two copies of the file, or two references to a single copy. However, in the different contexts or branches of the tree, the meaning and use of the source in one context is different from, and frequently independent of, the other context.

Files are frequently copied in order to support different contexts or usage, and administration is put in place to manage the copying act. The hierarchical views group and manage the various contexts, so that a developer can discover, through inspecting the hierarchical views, that a file exists in different contexts.

Tools exist that support a presentation in which a file item in one context represents the real location of the file, while all other contextual representations of that file are merely references to its real location. Using the example set forth in Figure 1, source files A 8 and B 10a could physically exist

together in a directory for the edit session 4, and the representation of source file B 10b under the build target 6 would then indicate a reference to the physical location of the source file B in the directory tailored to the edit context. Using a visual tool, the fact that source file B 10b under the build target 6 is a reference can be shown by colouring the line or node in the hierarchy in a contrasting colour, or with textual descriptive attributes of the line or target graphical node.

The value of hierarchies, then, is that they show objects in context and assist developers in managing the fact that an object may exist in many contexts. However, a main weakness of the hierarchies is that they are directly tied to physical locations. This means that administration is still required to deal with the physical location, with copying and with other functions associated with file system maintenance.

Tools currently in existence do permit some indirection in the views, but the views are still anchored in the physical locations. A useability issue that arises is that if colours are not used to differentiate between views of items representing actual physical locations and views of items representing references, it is never clear what a delete or update action means. That is, it is not clear whether deleting a view of an item will delete the physical source and thus all references to it, or merely a reference to a physical source file. The same concerns apply with respect to updating an original or a mirror copy of it.

The solution provided by the present is to cut the tie between the view or context and the physical location of the objects. Two separate containments are created, a logical view for all contextual views, and a physical view. This means that every item or object in a context can only have an indirect link to a physical location. All objects physically exist in only one location, and if a copy of an object is made, it is created as a new object.

This is illustrated schematically in Figure 2. In this tree view, in addition to the Project 14 hierarchy, there is an independent hierarchy called File Physical View 16. File Physical View 16 and the tree source files A 18, B 20 and C 22 it contains, represent the physical location of the directory

containing the actual source files.

The hierarchy of views under Project 14 represent pieces of the information required in the different task context of the edit session 24 and build target 26. The items or source files 28, 30, 32 and 34 listed in each view contain only indirect references to the corresponding real file objects 18, 20 and 22. This is shown by the broken lines linking these items to their respective context view roots. Thus, only a view becomes tailored to a particular context, rather than an underlying physical directory and location of the source file.

A single semantic can be applied to all visual references. For example, deletion or relocation of an item in a context view really modifies its reference to the physical location, rather than modifying the physical location of the actual source file.

The File Physical View 16 is constructed for explicit operations that deal with physical locations, such as copying and relocation.

The invention has been implemented in a tool builder framework having a metadata layering capacity to separate the information on logical views of the data in different contexts from the physical source code files. The aspects of this framework and metadata layering capacity are fully described in two concurrently filed application titles "An Object Oriented Framework Mechanism Providing Common Object Relationship and context Management For Multiple Tools" (IBM Docket No. CA997-002) and "Hierarchical Metadata Store For An Integrated Development Environment" (IBM Docket No. CA997-003), which are commonly assigned.

Figure 3 illustrates a window view of the software context management system according to a preferred implementation. In this window 36, Session1 38 is a visual representation of the hierarchy containing all of the physical source files with "car" data. These files include CarDatastore 40, CarManagerSequence 42, CarManager 44, CarManagerBase 46, Car 48 and CarDataId 50. A user can construct in a contextual Project1 directory 52, a compile unit in the form of a dynamic link

library for a Car 54 containing specific attributes of the car data from the physical source files. The items in Car.dll 54 are references to the physical source code files. Therefore, the logical item CarDatastore 56 is a reference to the actual file CarDatastore 40, CarManager 58 a reference to CarManager 44 and Car 60 a reference to Car 48. Preferably, the logical items are visually indicated, for example, by the use of contrasting colour.

Embodiments and modifications of the invention that would be obvious to a person skilled in the art are intended to be covered within the scope of the appended claims.

The embodiments of the invention in which an exclusive property or privilege is claimed are defined as follows:

1. In a file data processing system, a mechanism for managing source code files in file directory hierarchies, comprises:

a first hierarchy having at least one branch containing objects representing a physical location of the source code files; and

at least one additional hierarchy having branches representing contextual views of items from the source code files, said items being references to individual source code files.

2. A mechanism, according to claim 1, further comprising:

means for displaying said first hierarchy and said at least one additional hierarchy on a display associated with said file data processing system.

3. A mechanism, according to claim 2, wherein a user modification of an item displayed in one of said at least one additional hierarchies modifies the item's reference to an individual source code file, and a user modification of an object displayed in said first hierarchy modifies the object's physical source code file.

4. A mechanism, according to claim 3, wherein a user modification to copy a first object displayed in the first hierarchy creates a unique second object representing a physical copy of the first object's physical source code file.

5. In a file data processing system, a method for managing source code files in file directory hierarchies comprising:

maintaining a first hierarchy having at least one branch containing objects representing a physical location of the source code files; and

creating at least one additional hierarchy having branches representing contextual views of items from the source code files, said items being reference to individual source code files.

6. A method, according to claim 5, further comprising:

displaying said first hierarchy and said at least one additional hierarchy on a display associated with the file data processing system.

7. A method, according to claim 6, further comprising:

modifying a reference to an individual source code file in response to a user modification of an item containing the reference in an additional hierarchy on the display.

8. A method, according to claim 6, further comprising:

modifying a physical source code file in response to a user modification of an object representing the physical source code file in the first hierarchy on the display.

9. A method, according to claim 8, wherein the user modification of the object is a copy function, further comprising:

creating a new object in the first hierarchy representing a physical copy made of a physical source code file.

FIGURE 1

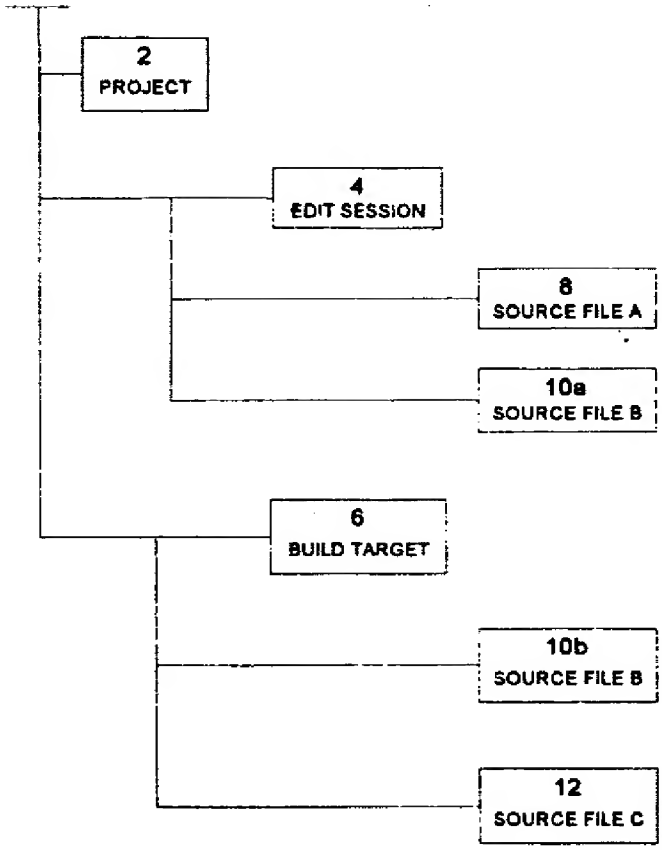

2
PROJECT
4
EDIT SESSION
8
SOURCE FILE A
10a
SOURCE FILE B
6
BUILD TARGET
10b
SOURCE FILE B
12
SOURCE FILE C

FIGURE 2

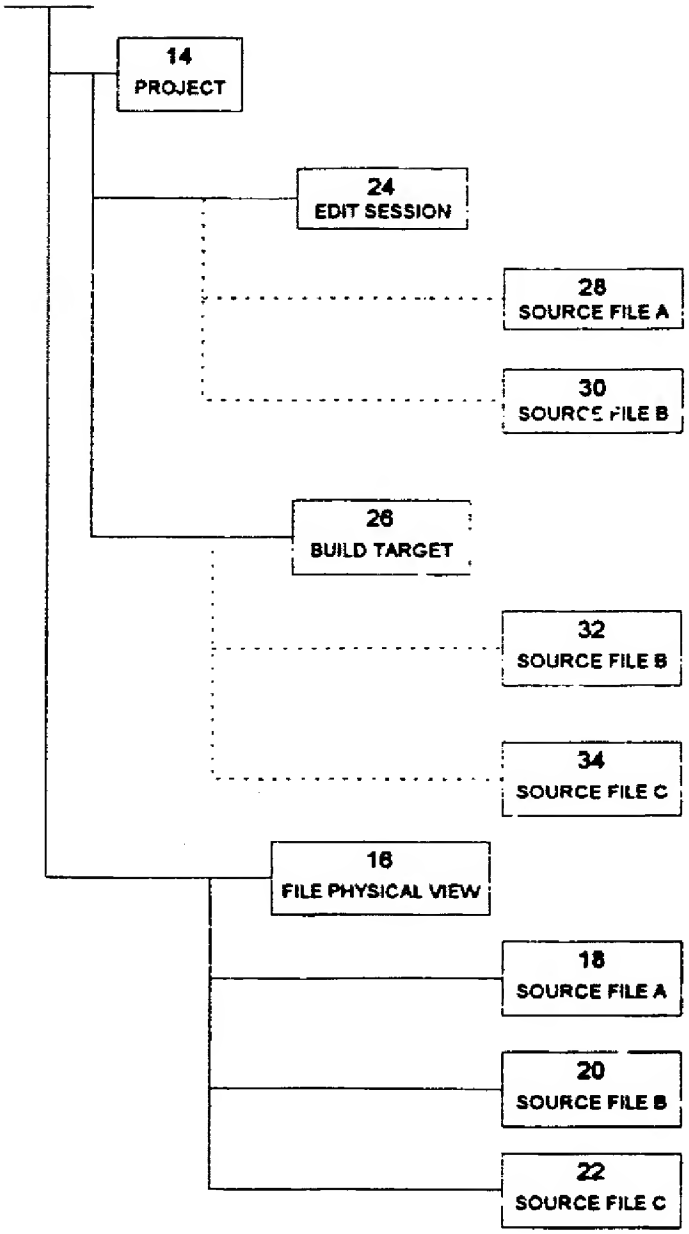


FIGURE 3